INTRODUCTION TO SILICON COMPILATION

J.P. Gray
California Institute of Technology
Pasadena, Calif. 91125

Inexorable progress in device scaling has given rise to obvious increases in circuit complexity. There is the conjecture that the level of complexity in hardware designs is akin to the level of complexity associated with large software systems. If this is the case, then it follows that the design methods and expertise of systems analysts could be brought to bear on the complexity problems associated with large designs in silicon. Already there is evidence that structured hardware design, analogous to structured programming, is emerging in design philosophies that emphasize wiring management and hierarchical design development with regular structures [1]. However, if the expertise of the personnel in the software world is to be applied to silicon implementations of systems then there must be mechanisms that allow their participation in the design process. This could most effectively be achieved by allowing them to write programs which, when compiled, yield code that produces manufacturing data for silicon parts. Thus, taking a macroscopic view, there is a need to provide design tools that take a completely textual description of a design and translate it to layout data.

On a more microscopic scale there is also an increasing necessity for program descriptions of sub-structures. This occurs when regular blocks, such as memories and PLA's, are programmed for specific functions. Thus, on two levels, there is a need for textual representation translation, or silicon compilation.

It may be worth comparing the silicon compilation task with the more conventional high level language compiler in order to set the present work in perspective and help identify the problem areas of the future. The use, and benefits, of high level languages and structured design methods for software systems is well documented elsewhere [2] [3]. There is also a history of languages used in design. These fall mainly into two categories: register transfer languages [4] and graphics languages [5]. In the first of these categories RTLs have to be used to capture behavioral descriptions of designs. By providing simulation, via compilation and execution of the RTL description, and physical design sub-systems it has been possible to construct hardware automatically, although at a cost in space and speed. In some recent work [6] it has been possible to compile a PDP-8 from an ISP behavioral description using standard modules with a chip count within 50% of a commercial design.

Graphic languages have a long history of use in design, especially for describing integrated circuit artwork. They are well suited to handling repetition for regular blocks, parameterisation for flexible scaling and hierarchy for structuring a design. We have seen the creation of graphics language for a number of objectives from design sharing [7] to providing an interface to manufacturing [8]. These are all however, physical descriptions.

Thus design languages have been used to capture, in varying degrees, the three descriptions key to hardware design: structural, behavioral and physical. To a first approximation this necessity for three developed descriptions to completely characterize a hardware design places significant demands on the discipline of the designer. The amount of work involved in the preparation of this data may be reduced by selecting an appropriate design philosphy that simplifies the relationships between the three hierarchical descriptions. The unification of the structural and physical hierarchies in the design style due to Mead [1] for example.

There remains the question of what constitutes a silicon compiler. Taking the simple minded view that a compiler translates a high level description to a low level description, then the use of a high level graphic language for producing artwork constitutes silicon compilation. On the other hand there is the view that hardware compilation in general takes a behavioral description of a system and maps it onto a physical structure. The work reported in this session is more closely aligned with the first of these definitions whilst work reported elsewhere in this conference follows the second of the definitions. There is likely to be much discussion on the costs and benefits of placing emphasis on a structural or behavioral approach to silicon compilation.

In this session there is a description of a extensible language system with associated programming environment and application of the system to two design tasks. The results show that structured designs can be described by structured programs and that data type extensions provides a method of putting together hierarchical descriptions. The benefits of parameterised specification is also clearly demonstrated in the task of chip assembly.

For the future we must clarify the role of behavioral descriptions in the design process. One view would treat them simply as apendages to structural data, generated from concentration on the important wiring management problems of large designs, so that verification by simulation can be carried out. Alternatively they may be viewed as the only desirable high level descriptions to be manipulated by designers who should not be concerned with the generation of data at lower levels of design abstraction.

References

[1] C. Mead & L. Conway
"Introduction to VLSI Systems"
Text in preparation

[2] N. Wirth
"Algorithm + Data Structures = Programs"
Prentice-Hall

[3] L.C. Jones, D.A. Nelson
"A Quantitative Assessment of IBM's Programming Productivity Techniques"
DA Conference Proceedings 1976

[4] M.R. Barbacci, G.E. Baener, R.G. Cattell, D.P. Siewiorek
"The ISPS Computer Description Language"
Technical Report, Computer Science Dept., Carnegie-Mellon Univ., Pittsburgh, Pennsylvania, 1977

[5] R. Ayres
"A Language Processor and Sample Language"
Silicon Structures Project Report #2276
California Institute of Technology, June 1978

[6] A.C. Parker
"Specification, Simulation and Automated Design of Interfaces and Digital Circuits"
Final Report to U.S. Army Reseach Office,
Carnegie-Mellon Univ., July 1978

[7] J. Eades
"GAELIC Users Manual"
Wolfson Institute, University of Edinburgh

[8] R. Sproull, R.F. Lyon
"The Caltech Intermediate Form for LSI Layout Description"
Display File Memo #1120, March 1979.